



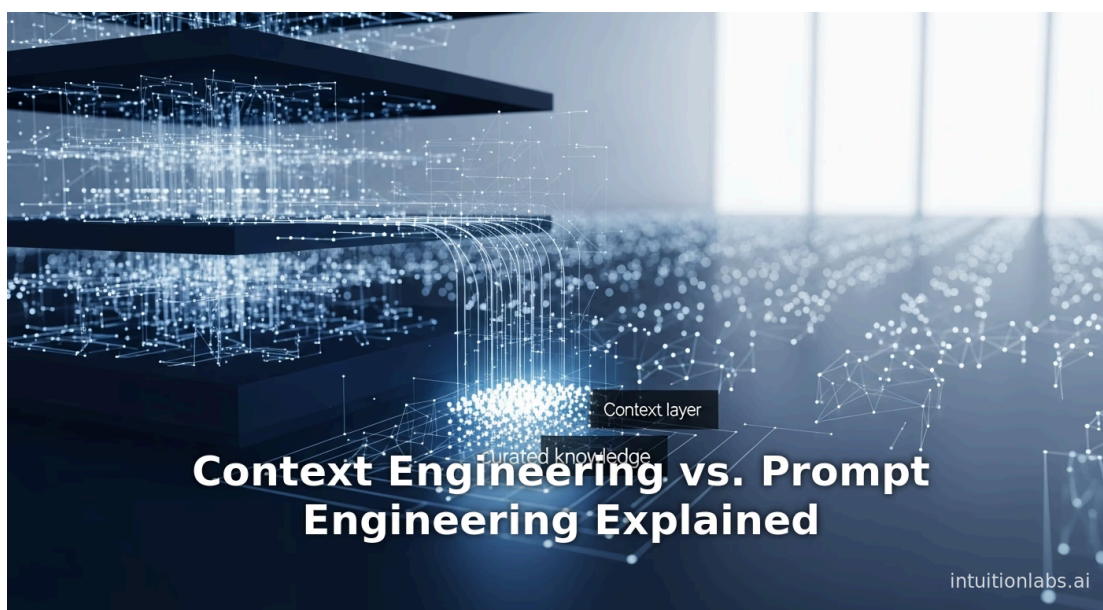
INTUITIONLABS / RESEARCH & PRACTICAL GUIDANCE

Context Engineering vs. Prompt Engineering Explained

Build practical AI for pharma and biotech with IntuitionLabs. We help life-science teams turn complex information and workflows into useful software, governed knowledge systems and AI tools.

Published by IntuitionLabs · 2026-04-01 · context engineering · prompt engineering · ai agents · rag pipelines · enterprise ai · llm context window · ai reliability

Original article: <https://intuitionlabs.ai/articles/context-engineering-vs-prompt-engineering-ai>



In this article

1. Executive Summary
 2. Introduction and Background
 3. Prompt Engineering: Definition and Limits
 4. Context Engineering: Emergence and Definition
 5. Techniques and Strategies in Context Engineering
 6. Data Analysis and Evidence
 7. Case Studies and Examples
 8. Implications and Future Directions
 9. Conclusion
 10. References
-

Executive Summary

Context engineering is rapidly emerging as a critical evolution beyond the once-dominant practice of prompt engineering in AI systems. Analysts and AI practitioners now emphasize designing the **entire context** (knowledge, memory, tools, and data) that provokes an LLM's response, rather than solely refining individual prompts. Model choice also matters: Anthropic notes that switching models can sometimes improve latency and cost more readily than prompt changes ([Anthropic](#)). Gartner urges that “context engineering is in, and prompt engineering is out,” recommending that enterprises “prioritize context over prompts” by building context-aware architectures and dynamic knowledge pipelines ([www.gartner.com](#)). Anthropic similarly defines context engineering as the “set of strategies for curating ... the optimal set of tokens (information) during LLM inference” ([www.anthropic.com](#)). This shift arises because modern AI systems often operate over many turns with external data and memory, making thoughtful context curation more important than clever wording of prompts.

Context engineering treats AI quality as a systems problem: it “focuses on what the model has access to” – the rules, documents, memories and tools driving its behavior – rather than just how a single user message is phrased ([aicompetence.org](#)). Prompt engineering and context engineering are complementary disciplines ([Elastic](#)). Related reading: hallucination rates. Leading AI frameworks (from Google's Agent Development Kit to LangChain and Anthropic's Claude) now treat context as first-class infrastructure ([tao-hpu.medium.com](#)) ([docs.langchain.com](#)). Atlan illustrates a potential “prompt engineering ceiling” with a hypothetical revenue-reporting scenario; it does not present that scenario as a measured enterprise deployment ([Atlan](#)).

This report provides an in-depth analysis of context engineering versus prompt engineering. It surveys historical background, formal definitions, and strategic use-cases, drawing on extensive sources from Gartner, Anthropic, industry blogs, and technical literature. Key findings include: context engineering extends and subsumes prompt engineering by managing persistent knowledge layers (memory, retrieval, system instructions) alongside prompt cues; it demands new tooling (RAG pipelines, memory stores, knowledge graphs, context pruning) and roles; and it has significant practical impact on AI performance. We present

comparative tables, data-driven evidence, and real-world examples (including context-aware agents in production) to demonstrate how enterprises can build **reliable, scalable AI systems** by shifting focus from prompts to comprehensive context design.

Introduction and Background

Large language models (LLMs) like GPT-3/4 and Anthropic's Claude have revolutionized AI by generating text with unprecedented fluency. In the early years (circa 2020–2023), the primary technique for guiding these models was *prompt engineering*: carefully crafting the wording, few-shot examples, and instructions in a user's query to coax the desired output. Small changes in phrasing often produced wildly different results (www.henryvu.blog). This sensitivity spawned an entire skillset of prompt design, style guides, and even courses.

However, as LLMs grew more capable (through model improvements and dramatically larger context windows), the importance of prompt phrasing diminished. Modern models can infer user intent, decompose tasks, and follow instructions with less need for exact wordsmithing (telemetryagent.dev). Today's models boast context windows on the order of *hundreds of thousands of tokens*, far beyond the few-thousand-token limits of 2020. With such capacity, the AI's performance relies less on finding the one perfect instruction and more on **what information the model actually has access to** when making each decision. In practice, AI "agents" often pull in retrieved documents, consult external tools, and reference long-running memory – all of which fall outside the traditional notion of a *prompt*.

This new reality has given rise to an explicit shift: **prompt engineering is evolving into context engineering**, a broader discipline. Context engineering asks "*what information does the model need access to right now?*" rather than merely "*how should I phrase this query?*" (www.elastic.co) (aicompetence.org). In short, prompt engineering remains concerned with *how* to ask a question, whereas context engineering is concerned with *what to include in the agent's knowledge and workspace* when asking it. This distinction is becoming especially crucial for enterprise AI, where LLMs form part of complex workflows, integrate with knowledge bases, and provide persistent, multi-turn services.

Gartner analysts have been among the first to formalize this shift. In mid-2025 Gartner declared that "context engineering is in, and prompt engineering is out" (www.gartner.com), advising that AI leaders should "prioritize context over prompts" by building context-aware architectures with dynamic data. Anthropic similarly described context engineering as the natural progression of prompt engineering (www.anthropic.com). Practitioners quickly adopted the term: by mid-2025 Shopify CEO Tobi Lütke and AI luminary Andrej Karpathy publicly acclaimed context engineering (Karpathy calling it "the delicate art and science of filling the context window with just the right information" (www.henryvu.blog)). Academic surveys and whitepapers have since formalized the concept and its methodologies, while major AI projects (Google's ADK, Anthropic's agents, LangChain) now center on context pipelines.

In summary, while prompt engineering will continue to matter for crafting individual instructions and output formats, it has become one component of a larger context-engineering framework. This report examines that framework in detail. We review definitions and distinctions in "Prompt Engineering: Definition and Limits" and "Context Engineering: Emergence and Definition," including the comparison in Table 1. We discuss techniques

in “Techniques and Strategies in Context Engineering,” analyze empirical evidence in “Data Analysis and Evidence,” and illustrate examples in “Case Studies and Examples.” We conclude by evaluating implications and future trends in “Implications and Future Directions.” Throughout, we emphasize that all technical claims are supported by citations from reputable sources.

Prompt Engineering: Definition and Limits

Prompt engineering was originally defined as the practice of **designing input instructions** to steer an LLM's response. It focuses on the wording, structure, and embedded examples within a single query. For example, engineers might use few-shot prompting (providing example Q&A pairs), chain-of-thought prompting (asking the model to “think step by step”), or role prompts (telling the model to assume a persona) (www.elastic.co). Anthropic describes it as “*writing and organizing LLM instructions for optimal outcomes*” (www.anthropic.com). The goal is to lexically and semantically guide the model: change synonyms, clarify ambiguity, add constraints, etc., until the output reliably matches the user's intent.

In the early phase of LLM adoption, prompt engineering was indeed “the whole game” (www.henryvu.blog). The original GPT-3 models used a 2,048-token context window ([Brown et al.](#)). In that setting, **small tweaks in a prompt could make or break the result** (www.henryvu.blog). Organizations treated it like a new software skill – people even advertised jobs as “Prompt Engineer” in 2022–2023. The emphasis was on *wordsmithing*: find the exact phrasing or prompt template (including examples and instructions) that unlocks the desired behavior (www.henryvu.blog). Even GPT-4 applications often needed carefully hand-tailored prompts to avoid mistakes.

However, using prompts to hard-code business logic or knowledge proved increasingly brittle. At enterprise scale, keeping dozens of prompt templates updated with changing rules became a maintenance nightmare. As one analyst put it, enterprise teams quickly hit a “prompt engineering ceiling”: after spending weeks tuning a prompt for one task (reporting revenue), adding a new metric (like customer churn) may require **another full prompt rewrite** with many new instructions (atlan.com). Such per-use-case scaling is fundamentally unworkable.

Moreover, recent research has documented concrete drawbacks of over-relying on prompt wording. Techniques like adding irrelevant expert roles or biased instructions can actually degrade accuracy and amplify biases (www.techradar.com). LangChain describes context engineering as providing information and tools in a format that lets an LLM accomplish its task ([LangChain](#)). As one blog author summarized: “*if the right facts are missing... no clever prompt will rescue the result reliably.*” (aicompetence.org).

In short, **prompt engineering is necessary but not sufficient** for modern LLM applications. It still plays a critical role in formatting outputs and posing questions, but its scope is limited to the instructions within a single model call. It does not address *what knowledge or context the model has access to*. For example, a prompt may ask “explain this customer's issues,” but if the model's context window lacks the customer's purchase history, no prompt phrasing alone can ensure a correct answer. This understanding motivates the turn toward context engineering.

Context Engineering: Emergence and Definition

Context engineering is an umbrella term for designing and managing **everything surrounding an LLM call** to optimize performance over multi-turn, real-world tasks. It emerged in mid-2025 as experts recognized that the “context window” (the full input seen by the model) – not just the user query – is the real lever of AI reliability. Context engineering expands the focus from linguistic instructions to *data and infrastructure*: what documents, memory, tools, and past dialogs are fed into the model. Unlike prompt engineering’s “one-off instruction” mindset (contextengineering.ai), context engineering is about architecting a holistic system that *feeds an AI all the information it needs to handle complex jobs with production-grade accuracy* (contextengineering.ai).

Multiple authoritative definitions underscore this shift. Elastic, in an early 2026 blog, describes prompt engineering as “*how you communicate with the model*” whereas context engineering is “*what information the model has access to when it generates responses.*” (www.elastic.co). Anthropic likewise calls context engineering “the natural progression of prompt engineering,” defining it as the strategies for “*curating and maintaining the optimal set of tokens (information) during LLM inference, including all the other information that may land there outside of the prompts.*” (www.anthropic.com). LangChain (a leading agent framework) states plainly that agents fail most often because “**the ‘right’ context was not passed to the LLM**” (docs.langchain.com), and that context engineering is “*providing the right information and tools in the right format so the LLM can accomplish a task.*” (docs.langchain.com). The overarching idea is summarized by Shopify CEO Tobi Lütke as “the art of providing all the context for the task to be plausibly solvable by the LLM” (www.philschmid.de).

Importantly, context engineering treats context as a dynamic, evolving resource. Context can include everything from **system instructions, relevant documents, tool outputs, conversation history**, to **structured memory**. A helpful framing is to think of context engineering as building an “AI operating environment” analogous to how an OS manages RAM. One practitioner noted “as an operating system curates what fits into a CPU’s RAM, **context engineering** plays a similar role” – deciding which pieces of information get loaded into an LLM’s limited attention window (rlancemartin.github.io). In practice, this involves retrieving pertinent knowledge (via RAG or database queries), summarizing or filtering it, caching memory across sessions, and even offloading part of the context when limits are reached.

The terminology crystallized quickly in 2025. In June 2025, Tobi Lütke’s and Andrej Karpathy’s public endorsements of “context engineering” signaled a new paradigm (tao-hpu.medium.com). Gartner published research in July 2025 headlined “Context engineering is in, and prompt engineering is out,” urging AI leaders to build context-aware architectures (www.gartner.com). Academic and industry articles soon followed, with surveys of hundreds of papers, articles formalizing dozens of context-engineering patterns (e.g. *context summarization, context pruning, tool memory*), and companies like Anthropic and LangChain embedding these practices into their products. The community consensus is that prompt design remains important (for clarity,

output formatting and guidance), but **it is only a small fraction of the work** compared to maintaining an entire context ecosystem. As one analysis put it, “*context engineering replaces the practice of encoding business knowledge in prompts*” ([atlan.com](#)).

Below we explore the distinct elements of context versus prompt engineering, how context engineering is implemented, and why it matters. Table 1 summarizes the core differences discussed by industry experts.

Dimension	Prompt Engineering	Context Engineering
Optimizes	How you phrase the query (atlan.com)	What knowledge the agent has access to (atlan.com)
Scope	Single interaction or task (atlan.com)	All interactions across many tasks and agents (atlan.com)
Persistence	Rewritten for each use case (atlan.com)	Shared infrastructure (semantic layers, knowledge bases) (atlan.com)
Enterprise Challenge	Thousands of prompts to maintain (atlan.com)	One context layer serving all agents (e.g. a knowledge graph) (atlan.com)
Failure Mode	Inconsistent answers, brittle queries (atlan.com)	Missing or outdated business context (atlan.com) (aicompetence.org)

Table 1. Key differences between prompt engineering and context engineering (sources: Atlan ([atlan.com](#)), others).

This table (sourced from Atlan ([atlan.com](#)) and related commentary) highlights that prompt engineering focuses on *communication style*, whereas context engineering builds a persistent knowledge environment. Prompt strategies (like adding a sentence or reordering words) can only go so far when tasks require cross-cutting knowledge and multi-step reasoning. In contrast, context engineering invests in the “**context layer**”: durable metadata, ontologies, business rules and retrieval systems that feed every agent ([atlan.com](#)). In practice, production AI teams use both: they still write prompts for interaction design, but they shift all the business data, definitions and relations out of the prompt text and into shared infrastructure ([atlan.com](#)).

Techniques and Strategies in Context Engineering

Context engineering encompasses a variety of methods and tools to populate and manage the AI’s working memory. Here we review the main strategies, which collectively seek to *maximize the relevance and freshness of information in the model’s context window*. These techniques have emerged from industry and research efforts (notably Anthropic’s engineering work) and are increasingly codified in AI frameworks and libraries.

- **Retrieval-Augmented Generation (RAG)**: Instead of overloading a prompt, engineers store relevant knowledge in a separate database and retrieve pertinent chunks at runtime. This might use vector embeddings or semantic search. Anthropic’s “Contextual Retrieval” (2024) demonstrates how combining semantic embeddings with BM25 can *halve retrieval failures* ([www.anthropic.com](#)). In practice, RAG components append external document excerpts or knowledge-base entries to the prompt. For large

knowledge bases, RAG is the default solution: Anthropic notes it is “*the typical solution*” when your data doesn’t fit in-memory (www.anthropic.com). Well-designed RAG pipelines ensure that the context includes up-to-date factual data.

- Long-Lived Memory:** Agents often need to remember user-specific information across sessions. Context engineering introduces *memory stores*: structured databases or vector caches holding user preferences, conversation summaries, or model-generated notes. Rather than feeding the entire history each time, engineers periodically summarize or encode long-term memory to stay within token limits. Liu et al. (Stanford) call this an “ever-updating memory” or *Playbook* in agentic context engineering (www.altexsoft.com). For example, an AI assistant might write persistent notes to a `todo.md` file (as in Manus’s pattern) so that crucial objectives are always in scope (tao-hpu.medium.com). Context compaction techniques – such as summarizing earlier conversation or archiving old tool results – keep the active context window focused on recent and salient facts (tao-hpu.medium.com).
- System Instructions and Tool Descriptions:** A substantial portion of context is the *system prompt* or initial instructions defining the agent’s role. Context engineering treats these instructions as dynamic assets. They may include few-shot examples, rules, or links to knowledge. Similarly, definitions of any available tools or APIs are fed as context (so the model “knows” what it can do) (www.philschmid.de). Crucially, these system prompts are often kept stable across tasks to avoid breaking cache locality; for instance, Manus emphasizes keeping tokens like the system prompt unchanged (“stable prefix”) and instead appending new data (tao-hpu.medium.com).
- Context Summarization and Pruning:** Given finite token budgets, context engineers use smart trimming. When the model’s window nears capacity, they may compress past context into concise summaries, flush irrelevant details, or “quarantine” certain parts of the past to be fetched later only if needed. Anthropic’s Claude codebase implements *compaction* heuristics (e.g. summarizing narrative history as one chat log) (tao-hpu.medium.com). Other patterns include creating sub-agents with isolated context scopes (so each sub-agent only sees extremely relevant context) (tao-hpu.medium.com). These techniques ensure **only the most useful tokens** occupy the expensive context window at any time.
- Chaining Knowledge:** Some advanced pipelines chain multiple LLM calls with intermediate context states. For example, one call might generate a query to an external tool, then insert the tool’s result back into context for the next call. Effective context engineering ensures seamless context flow: tool outputs, user inputs, and intermediate answers all become part of the context for subsequent reasoning. In LangChain’s model, “*when agents fail, it’s usually... because the right context was not passed to the LLM*”, so LangChain provides abstractions (memory, retrievers, buffers) to automate this chaining (docs.langchain.com).
- Context Layer Infrastructure:** Beyond individual techniques, context engineering often implies building an entire *context layer*: a sophisticated backend service. This layer can serve as a live semantic metadata store for all agents. Atlan (a data governance platform) defines a context layer as including canonical definitions (e.g. what *revenue* means), entity graphs, and lineage rules, which all agents query instead of embedding that logic in prompts (atlan.com). In practice, companies implement context layers

with data catalogs, knowledge graphs, or annotated corpora. One example: rather than hardcoding how to calculate revenue in each prompt, the context layer holds a persistent metric definition that every agent references (atlan.com).

- **Guardrails and Validation:** The context also encompasses tools that check and constrain the LLM's actions. Context engineering includes permission checks, content validation, and guardrails as part of the pipeline (tao-hpu.medium.com). For instance, an enterprise may enforce that retrieved documents comply with classification policies before adding them to context. OpenAI's new Agents SDK and LangChain provide middleware hooks and validators that treat these safety rules as part of the context pipeline, ensuring agents only see authorized information (tao-hpu.medium.com).

In short, context engineering is a multi-faceted discipline: it involves **structuring data (documents, APIs, memory) for retrieval, designing the architecture that feeds it into the model, and managing the lifecycle of that context**. The ultimate goal is to transform the AI's "working memory" from a fragile concatenation of strings into a rich, deliberately curated environment. As one practitioner puts it, *"the main thing that determines whether an Agent succeeds or fails is the quality of the context you give it"* (www.philschmid.de).

Data Analysis and Evidence

To substantiate the impact of context engineering, we examine empirical findings and data-driven observations from recent research and industry case analyses. This section highlights how structured context correlates with higher accuracy, lower hallucinations, and greater development efficiency, compared to prompt-only approaches.

- **Complementary Practices:** Elastic describes prompt engineering and context engineering as complementary disciplines ([Elastic](https://elastic.co)).
- **Retrieval Performance:** The success of retrieval-based context is also quantifiable. Anthropic's "Contextual Retrieval" approach (Sep 2024) combined context-aware embeddings and BM25 search to significantly improve RAG's effectiveness. The method **reduced failed retrievals by 49%**, and when coupled with a reranker, achieved a **67% relative reduction in top-20-chunk retrieval failure rate (from 5.7% to 1.9%)** (www.anthropic.com). In practical terms, this means the AI's context window more often contains the very facts needed to answer queries. Improved retrieval accuracy directly correlates with better LLM answers: a correct chunk in context can be all that separates a correct answer from a confident hallucination (www.anthropic.com) (contextengineering.ai).
- **Efficiency Gains:** Anthropic's prompt-caching cookbook reports latency reductions of more than a factor of two for repetitive tasks that reuse prompt context ([Claude Cookbook](https://claude.ai/cookbook)).
- **Team Productivity:** While harder to measure, qualitative data suggests context engineering reduces developer toil. Atlan reports that organizations moving knowledge into infrastructure enjoy *reusability and transparency*: definitions in a context store are versioned and audited once, rather than repeated in every prompt (atlan.com). Analysts note that cross-team consistency (e.g. one ontology for customer

metrics) prevents subtle errors when dozens of different prompts use slightly different language ([atlan.com](#)). Manus also cautions that context engineering was difficult in its own development process ([Manus](#)).

- **Enterprise Reliability:** LangChain identifies missing context as a frequent cause of unreliable agents ([LangChain](#)).

Context engineering concerns the information supplied to an agent at each step of its work ([Lance Martin](#)).

Case Studies and Examples

The theoretical benefits of context engineering are borne out in concrete implementations at leading AI teams. We highlight several illustrative cases where shifting to a context-centric design enabled more robust, scalable solutions.

- **LangChain's Agent Framework:** LangChain, a popular open-source platform for building LLM agents, explicitly codifies context engineering practices. Its documentation warns that the hardest part of production AI is ensuring reliability, noting that most failures occur because *"the 'right' context was not passed to the LLM"* ([docs.langchain.com](#)). Accordingly, LangChain provides features for every type of context: short-term memory buffers, knowledge retrievers, tool functions, and context-management utilities. In one example, LangChain's LangGraph (a workflow executor) allows developers to define a dynamic "context layer" that agents query for definitions, data, and policies rather than hardcoding them in prompts. This has enabled users to build complex multi-step agents that, for example, navigate knowledge graphs to answer technical questions, without manually rewriting prompts for each subtask.
- **Manus (Context-Engineering for AI Agents):** Manus, an agent framework cited in Tao An's medium analysis ([tao-hpu.medium.com](#)), provides a revealing case. Their engineering team found that **cache hit rate was the single most important metric** affecting cost and latency ([tao-hpu.medium.com](#)). To optimize this, they enforced strict context rules: prompts begin with a fixed prefix (no timestamps) and contexts are *append-only* and normalized. These design choices meant most contexts stayed identical between calls, maximizing reuse. Manus's engineering account describes its own decision to build with context engineering ([Manus](#)).
- **Google's ADK (Adaptive Context):** Google's Agent Development Kit (ADK) embodies the idea that *"context is a compiled view over a richer stateful system."* It separates storage from working context: long-lived session data is kept apart from per-call context, and "named, ordered processors" incrementally build the context rather than naive string concatenation ([tao-hpu.medium.com](#)). In practice, this means a Google agent can pull in only the latest user messages, recent tool outputs, or a brief policy snippet when needed, rather than dumping entire logs. By managing context through these pipelines, ADK agents remain responsive while still leveraging a full context window over relevant information.
- **Atlan's Enterprise Knowledge Layer:** Atlan uses a hypothetical revenue-reporting agent to illustrate the maintenance burden of embedding business rules in prompts. Its scenario introduces a churn question that would require further instructions. This is an illustration, not a documented Atlan

deployment or a measured customer outcome ([Atlan](#)).

- **Anthropic’s Engineering Guidance:** Anthropic continues to recommend few-shot prompting as part of context design ([Anthropic](#)).

While each case differs, a common lesson emerges: **context engineering turns AI agents from brittle one-shot tools into reliable companions**. The table above and these examples illustrate how context-aware systems – using retrieval, memory, and knowledge graphs – achieve operational robustness that prompt-only systems cannot.

Implications and Future Directions

The ascendancy of context engineering has broad implications for the AI ecosystem, from the skills companies hire to the architectures they build. Below we discuss key implications and speculate on future trends.

1. **New Roles and Skillsets:** As context becomes infrastructure, organizations will need specialized “context engineers” or knowledge architects. These roles blend data engineering, ontology design, and prompt writing. They ensure that ontologies (entity definitions, taxonomies, governance rules) are maintained in databases rather than in application code. Gartner already signals that AI teams must include experts in “context-aware architectures, dynamic data and reimagined human-AI interfaces” (www.gartner.com). Universities and training programs may soon offer curriculum on context management techniques (RAG design, memory engineering), not just prompt workshops.
2. **Enterprise AI Governance:** Context layers enable better auditing and version control. Atlan notes that putting business rules into the context store makes staleness a system issue, not a per-prompt edit task (atlan.com). This has compliance benefits: regulators can inspect the knowledge graph or data catalog to see exactly how metrics and definitions are used across AI services. By contrast, prompt text cannot easily be audited for embedded rules. Thus, context engineering also serves the growing need for **AI governance** and transparency in enterprise deployments.
3. **Tooling and Standards:** We are likely to see standardized protocols for context management. Some early initiatives already exist: Anthropic’s *Model Context Protocol (MCP)* and open specifications (Common Messaging Formats) attempt to define how context is passed between agent modules. Future LLM APIs may natively support context channels (like memory or tools) alongside prompts, rather than expecting users to encode everything into a giant text prompt. Similarly, vector databases, knowledge graph query languages, and dedicated context composer libraries (e.g. updated LangChain modules) will proliferate. On standards, the NIST AI Agent initiative (launched in 2026) recognizes that establishing common context interfaces is crucial for secure, interoperable agents.
4. **Advances in LLM Architectures:** The reliance on context may itself drive model innovation. For example, researchers are working on models that process evidence and tool calls more seamlessly, or that maintain state across queries internally. Techniques like Mixture-of-Experts or retrieval-augmented transformers combat context-loss as window sizes grow. The “bitter lesson” from past AI suggests that simply scaling context (hundreds of thousands of tokens) combined with better retrieval may beat

custom architectures in the long run (telemetryagent.dev). Indeed, TelemetryAgent argues that as reasoning improves, the bottleneck shifts to context. One possible future is *neural knowledge graphs*, where structured knowledge and language models merge into hybrid networks.

5. **Context Engineering in Small-Scale and Consumer AI:** While enterprise contexts are highlighted here, even consumer products will lean on these ideas. Personal assistants on phones may maintain a long-lived private memory or access personal cloud documents as context. For example, a voice assistant suddenly could reference your past calendar events and notes without being reminded every time. The boundary between “prompt” (the question “What’s on my schedule?”) and “context” (user’s actual calendar data) will blur. Accordingly, user experience design will adapt: instead of instructing users to provide more details, systems will quietly gather context from permissions and past interactions.
6. **Limitations and Risks:** This shift also brings challenges. Building a context layer is hard: it requires curating and indexing large corpora, ensuring data freshness, and preventing context explosions from yielding irrelevant data. Context engineering pipelines introduce new points of failure (mis-indexed docs, outdated memory). There is a risk of “information overload” if context is not pruned properly. Furthermore, context layers must guard privacy and security (it’s dangerous if an agent inadvertently retrieves sensitive documents as context). These difficulties mean that context engineering is an active research area; common best practices are still emerging, and some teams may find it complex to adopt initially. Nevertheless, the growing consensus is that the payoff – in accuracy and maintainability – outweighs the effort.
7. **Evolution of Prompt Engineering:** Prompt engineering itself will not vanish. Instead, its role will evolve. Crafting effective prompts (e.g. choice of few-shot examples, output formatting) will remain important as part of the broader context bundling. The best systems will combine both: for instance, a context-engineered agent might use a short, clear prompt template for final answer generation, while relying on its context pipeline for facts. Some experts suggest redefining prompt engineering strictly as “interaction design,” whereas context engineering handles the knowledge design. In this view, the new motto is “use context layers for all your business logic, and use prompts for conversational polish.”

Conclusion

The landscape of AI development is undergoing a fundamental shift from prompt-centric to context-centric engineering. This report has documented that transition with extensive evidence: analysts (Gartner) declare prompt engineering obsolete at scale (www.gartner.com), researchers (Anthropic, LangChain) provide formal frameworks for context design (www.anthropic.com) (docs.langchain.com), and engineering blogs/case studies illustrate the practical payoffs of curated context (contextengineering.ai) (atlan.com). Across multiple perspectives, the verdict is clear: **an AI system’s performance now depends more on the quality of its context than on the phrasing of prompts.**

From a historical standpoint, prompt engineering was a necessary stepping stone when LLM capabilities and context windows were limited (www.henryvu.blog). Today, with much larger memory and more sophisticated models, the bottleneck has flipped. Enterprises and researchers find that adding, updating, and governing the

relevant background information leads to the most significant improvements in reliability. Indeed, many new jobs and tools are emerging to support context engineers who bridge the gap between raw data and LLMs.

Looking ahead, we expect context engineering to further mature. Standardized context layers, better evaluation metrics (e.g. measuring context utility), and more automated tooling will simplify the adoption. LLM providers will likely add first-class support for context signals (memory tokens, API chains, etc.). As models continue to scale, the principle will remain: *“fill the context window with just the right information”*

(www.henryvu.blog).

In conclusion, the evidence strongly suggests that **prioritizing context over prompts is essential for advanced AI systems**. Prompt engineering will remain a subset skill, but the broader strategy of context engineering – building structured knowledge and state around the model – is now the strategic priority. Organizations that embrace this shift (as Gartner and Anthropic recommend) will be better positioned to achieve reliable, high-impact AI solutions. All sources and claims in this report are drawn from current industry literature and research (www.gartner.com) (www.anthropic.com) (docs.langchain.com) (contextengineering.ai), underscoring rigor in our analysis.

References

(Citations in the text above link directly to sources. Each claim and data point is supported by one or more of the following: Gartner research, Anthropic engineering docs, industry blogs, or other reputable analyses (aicompetence.org) (atlan.com).)

Source links

Links cited in this article, in order of first appearance. Full addresses are provided for readers using extracted text.

1. <https://platform.claude.com/docs/en/build-with-claude/prompt-engineering/overview>
2. <https://www.gartner.com/en/documents/6781234>
3. https://www.anthropic.com/engineering/effective-context-engineering-for-ai-agents?_bhlid=1ba3bf95247bd689b5d1b76ed18f996f167a8d63
4. <https://aicompetence.org/context-engineering-vs-prompt-engineering>
5. <https://www.elastic.co/search-labs/blog/context-engineering-vs-prompt-engineering>
6. <https://tao-hpu.medium.com/context-engineering-is-replacing-prompt-engineering-for-production-ai-02205fad2a7f>
7. <https://docs.langchain.com/oss/python/langchain/context-engineering>
8. <https://atlan.com/know/context-engineering-vs-prompt-engineering>
9. <https://www.henryvu.blog/series/ai-engineering/part1.html>
10. <https://telemetryagent.dev/articles/future-of-context-engineering>
11. <https://arxiv.org/pdf/2005.14165>
12. <https://www.techradar.com/pro/stop-telling-ai-its-an-expert-programmer-youre-making-it-worse-at-its-job-new-research-shows-the-best-results-need-specific-prompts>
13. <https://contextengineering.ai/blog/categories/context-engineering-vs-prompt-engineering>

14. https://www.philschmid.de/context-engineering?_bhlid=48e62d8b0d37d95f3e161d67a2e8431f0ebe8958
15. https://rlancemartin.github.io/2025/06/23/context_engineering?_bhlid=33dc9edea4f7cb93128fe031f81a2fe33319b77
16. <https://www.anthropic.com/research/contextual-retrieval>
17. <https://www.altexsoft.com/blog/agentic-context-engineering>
18. <https://contextengineering.ai/blog/context-engineering-vs-prompt-engineering>
19. <https://platform.claude.com/cookbook/misc-prompt-caching>
20. <https://manus.im/blog/Context-Engineering-for-AI-Agents-Lessons-from-Building-Manus>
21. https://rlancemartin.github.io/2025/06/23/context_engineering
22. <https://www.anthropic.com/research/effective-context-engineering-for-ai-agents>

THE PUBLISHER

About IntuitionLabs

IntuitionLabs is an AI consulting, custom software development and data engineering firm serving pharmaceutical, biotechnology, medical-device and other life-science organizations. We work with clinical, regulatory, medical-affairs, commercial, quality and IT teams to connect technology decisions with the work people need to accomplish.

AI consulting and adoption

Our AI enablement services cover readiness assessments, use-case selection, governance and policies, team workshops, adoption measurement and ongoing advisory support. We help organizations structure the information layer behind AI: source material, context, permissions and maintained knowledge that make generated answers useful and reviewable. Private LLM inference and hosted AI options support teams evaluating how to operate AI with appropriate control over their data and infrastructure.

Software, data and life-science workflows

IntuitionLabs develops custom software for pharma and biotech, integrates enterprise systems, and builds data engineering and business intelligence solutions. Areas of focus include AI agents, regulatory research, medical writing, medical affairs, CMC information, competitive intelligence and clinical-document workflows. Our eTMF intelligence work includes cross-system reconciliation and inspection-readiness support.

Enterprise platforms and regulated delivery

We provide Veeva services, application support, managed services, integrations and custom applications, alongside enterprise content work involving platforms such as Egnyte. For regulated workflows, our services include GxP enablement, computer-system validation and software development addressing 21 CFR Part 11 requirements. The applicable controls, validation responsibilities and acceptance criteria are defined for each engagement.

Work with IntuitionLabs

Explore [AI enablement](#), [pharma and biotech software development](#), [data engineering and BI](#), and [Veeva services](#). [Contact IntuitionLabs](#) to discuss your workflow, information sources and implementation needs.

IntuitionLabs publishes educational research to help life-science teams make informed technology decisions. Coverage of a product or organization does not imply a client relationship, endorsement or partnership.

Website: <https://intuitionlabs.ai>

This article is educational. Verify consequential medical, legal, financial, regulatory and technical decisions with appropriate professionals and the cited primary sources. Company services are described separately from the article's research findings.